UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/711,148	08/27/2004	Allen Bauer	BORL/0221.01	5147

28653 7590 09/11/2007
JOHN A. SMART
708 BLOSSOM HILL RD., #201
LOS GATOS, CA 95032

EXAMINER	
WANG, BEN C	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
09/11/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/711,148	BAUER ET AL.	
	Examiner	Art Unit	
	Ben C. Wang	2192	

-- *The MAILING DATE of this communication appears on the cover sheet with the correspondence address* --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 August 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-48* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-48* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *10/12/2006*.	6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-48 are pending in this application and presented for examination.

Specification Objections

2. The specification is objected to because the following informalities:

- "IBM-PC" and "Microsoft Windows", e.g., cited in [Para 3], Line 4, are registered trademarks.
- "Java" and "C#", e.g., cited in [Para 10], Line 4, are registered trademarks.
- "Novell", e.g., cited in [Para 14], Line 17, is registered trademark.
- "Macintosh", "Linux", and "Solaris", e.g., cited in [Para 36], Line 10, are registered trademarks.
- "Intel", e.g., cited in [Para 39], Line 9, is registered trademark.
- "Hewlett Packard", e.g., cited in [Para 42], Line 10, is registered trademark.
- "Dell", "Apple", e.g., cited in [Para 44], Lines 2, 4 respectively, are registered trademarks.
- "Sun Microsystems", e.g., cited in [Para 47], Line 6, is registered trademark.
- "Microsoft Windows 9x", "Microsoft Windows NT", "Microsoft Windows 2000", and "Microsoft Windows XP", e.g., cited in [Para 36], Lines 11-12, are registered trademarks.

Appropriate correction is required (See MPEP § 608.01(b))

Claim Objections

3. Claims 2, 28, and 54 are objected to because the following informalities:

- "components that may placed on the particular form.", claim 29, line 3, should be corrected "components that may be placed on the particular form."

Appropriate correction is required.

Claim Rejections – 35 USC § 101

35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

4. Claims 25-48 are rejected under 35 U.S.C 101 because the claims are directed to non-statutory subject matter.

5. **As to claim 25**, "an ancestor class", "a proxy module", "a type delegator", and "a module for displaying the particular form", are being cited; however, it appears that the ancestor class, the proxy module, the type delegator, and the module for displaying the particular form would reasonably be interpreted by one of ordinary skill in the art as computer listings per se, are not physical "things". They are neither computer components nor statutory processes, as they are not "act" being performed. Such claimed computer programs do not define any structural and functional interrelationships between the computer program and

other claimed elements of a computer which permit the computer program's functionality to be realized. In contrast, a claimed computer readable medium encoded with a computer program is a computer element which defines structural and functional interrelationships between the computer program and the rest of the computer which permit the computer program's functionality to be realized, and is thus statutory. Accordingly, it is important to distinguish claims that define descriptive material per se from claims that define statutory inventions. (See MPEP 2106.01(I))

6. **As to claims** 26-48, they are merely further recited as the computer program product per se, thus, do not cure the deficiency of base claim 25 above, and also rejected under 35 U.S.C. 101 as set forth above.

Claim Rejections – 35 USC § 112

The following is a quotation of the second paragraph of 35 U.S.C. 112:
The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

7. Claims 2 and 26 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

8. **Claims 2** and **26** contain the trademark/trade name of "Microsoft .NET Framework". Where a trademark or trade name is used in a claim as a limitation to identify or describe a particular material or product, the claim does not comply with the requirements of 35 U.S.C. 112, second paragraph.

See Ex parte Simpson, 218 USPQ 1020 (Bd. App. 1982). The claim scope is uncertain since the trademark or trade name cannot be used properly to identify any particular material or product. A trademark or trade name is used to identify a source of goods, and not the goods themselves. Thus, a trademark or trade name does not identify or describe the goods associated with the trademark or trade name. In the present case, the trademark/trade name is used to identify/describe "Microsoft .NET Framework" and, accordingly, the identification/description is indefinite.

9. **As to claims 2** and **26**, recite the term "Microsoft .NET Framework" that is a relative term and renders the claim indefinite. The term " Microsoft .NET Framework " is not defined by the claim, the specification does not provide a standard for ascertaining the requisite degree, and one of ordinary skill in the art would not be reasonably apprised of the scope of the invention.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

10. Claims 1-7 and 12-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lam et al., (*.NET® Framework Essentials, June 2001, O'Reilly®*) (hereinafter 'Lam') in view of Albahari et al., (*C#® in a Nutshell, 2nd Edition, August 2003, O'Reilly®*) (hereinafter 'Albahari')

11. **As to claim 1**, Lam discloses in a form-based development system (e.g., Chapter 8 – Windows Forms, Lines 4-6 – how to use Windows Forms .NET® classes to create Windows® Forms-based applications), a method for dynamically constructing a form under an object framework during development of an application by a user, the method comprising:

- providing an ancestor class under an object framework, the ancestor class for representing a form in the development system; in response to user input, creating a descendant class inheriting from the ancestor class for representing a particular form to be included in the application (e.g., Fig. 8-2 – System.Windows.Forms Windows Controls Class Hierarchy; Sec. 8.2 – The System.Windows.Forms Namesapce, 3rd Para. – the System.Windows.Forms namespace provides common set of classes you can use and derive from to build Windows® Forms application; the classes and interfaces in this namespace allow you to construct and render the user-interface elements on a Windows® Form);

- generating instructions for creating methods of the descendant class under the object framework; creating an instance of the descendant class; and constructing the particular form in the development system based on the instance of the descendant class (Sec. 8.3.3 – Visual Inheritance, 2nd Par. – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes).

Lam does not explicitly disclose creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application.

However, in an analogous art of C#® in a Nutshell, Albahari discloses creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application (e.g., Fig. 35-2 – Exceptions, delegates, and attributes from System.Reflection – element of “TypeDelegator”; Chapter 35. – System.Reflection, 1st Par. – System.Reflection is the API that exposes the full-fidelity metadata of the .NET environment to the programmer; In short, it permits complete access to compile-time data at runtime; Everything is available including fields, methods, constructors, properties, delegate types, and events).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Albahari into the Lam's system to further provide creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application in Lam system.

The motivation is that it would further enhance the Lam's system by taking, advancing and/or incorporating Albahari's system which offers significant advantages that reflection offers up a number of possible approaches to user; introspection is the act of using the reflection APIs to discover information about a component assembly (and its constituent types) at runtime without any prior (compile-time) knowledge of it as once suggested by Albahari (e.g., Chapter 35. – System.Refection, 2nd Par.).

12. **As to claim 2** (incorporating the rejection in claim 1), Lam discloses the method wherein the object framework includes Microsoft .NET® Framework (Sec. 8.3.3 – Visual Inheritance, 2nd Par. – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes).

13. **As to claim 3** (incorporating the rejection in claim 1), Lam discloses the method wherein said creating step includes creating a descendant class for representing the particular form in a user interface of the development system (e.g., Fig. 8-2 – System.Windows.Forms Windows Controls Class Hierarchy;

Sec. 8.2 – The System.Windows.Forms Namesapce, 3rd Para. – the System.Windows.Forms namespace provides common set of classes you can use and derive from to build Windows® Forms application; the classes and interfaces in this namespace allow you to construct and render the user-interface elements on a Windows® Form).

14. **As to claim 4** (incorporating the rejection in claim 1), Lam discloses the method wherein said creating step includes inheriting a set of components provided by the ancestor class for representing components that may placed on the particular form (e.g., Fig. 8-2 – System.Windows.Forms Windows Controls Class Hierarchy; Sec. 8.2 – The System.Windows.Forms Namesapce, 3rd Para. – the System.Windows.Forms namespace provides common set of classes you can use and derive from to build Windows® Forms application; the classes and interfaces in this namespace allow you to construct and render the user-interface elements on a Windows® Form).

15. **As to claim 5** (incorporating the rejection in claim 5), Lam discloses the method wherein said creating step includes creating an assembly for the descendant class (e.g., Sec. 2.4.1 – Assemblies Versus Components – In .NET®, Microsoft® has addressed this confusion by introducing a new concept, assembly, which is a software component that supports plug-and-play, much like a hardware component).

16. **As to claim 6** (incorporating the rejection in claim 1), Lam discloses the method further comprising: creating a second descendant class which inherits from the descendant class, the created second descendant class for representing a form which inherits from the particular form (Sec. 8.3.3 – Visual Inheritance, 2nd Par. – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes).

17. **As to claim 7** (incorporating the rejection in claim 1), Lam discloses the method wherein said constructing step includes constructing the particular form based upon the descendant class in a user interface of the development system (Sec. 8.3.3 – Visual Inheritance, 2nd Par. – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes).

18. **As to claim 12** (incorporating the rejection in claim 1), Albahari discloses the method wherein said generating step includes generating a constructor for the descendant class (e.g., Chapter 35 – System.Reflection – everything is

available including fields, methods, constructor, properties, delegate types, and events).

19. **As to claim 13** (incorporating the rejection in claim 12), Lam discloses the method wherein said step of generating a constructor includes generating intermediate language instructions for building the constructor (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be converted into IL, so .NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR]).

20. **As to claim 14** (incorporating the rejection in claim 13), Lam discloses the method wherein said step of generating intermediate language instructions includes using classes provided by the object framework for generating intermediate language instructions constructor (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be

converted into IL, so .NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR]).

21. **As to claim 15** (incorporating the rejection in claim 13), Lam discloses the method wherein said generating step includes generating instructions for calling the constructor of the ancestor class, thereby ensuring execution of an appropriate constructor implemented by the ancestor class (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be converted into IL, so .NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR]).

22. **As to claim 16** (incorporating the rejection in claim 1), Lam discloses the method wherein said generating step includes generating methods for overriding notification methods of the ancestor class (e.g., Sec. 8.2.2.1 – Extending existing controls – because Windows® Forms API is object oriented, extending controls is as easy as deriving from the default behavior of the control).

23. **As to claim 17** (incorporating the rejection in claim 16), Lam discloses the method wherein said generating step includes generating intermediate language

instructions for overriding notification methods of the ancestor class (e.g., Sec. 8.2.2.1 – Extending existing controls – because Windows® Forms API is object oriented, extending controls is as easy as deriving from the default behavior of the control).

24. **As to claim 18** (incorporating the rejection in claim 1), Albahari discloses the method wherein the type delegator provides information for enumerating fields, methods, properties, and events in response to user input on the particular form in the development system (e.g., Chapter 35 – System.Reflection, 1st Para. – everything is available including fields, methods, constructors, properties, delegate types, and events).

25. **As to claim 19** (incorporating the rejection in claim 1), Lam discloses the method wherein the type delegator generates metadata information in response to user input on the particular form (e.g., Sec. 2.3 – Metadata, 1st Para. – Metadata is machine-readable information about a resource, or "data about data"; Such information might include details on content, format, size, or other characteristics of a data source; In .NET®, metadata includes type definitions, version information, external assembly references, and other standardized information; Sec. 2.3.1, 1st Para. – Just as type libraries are C++ header files on steroids, metadata is a type library on steroids; in .NET® metadata is a common mechanism or dialect that the .NET® runtime, compilers, and tools can all use. Microsoft .NET® uses metadata to describe all types that are used and exposed

by a particular .NET® assembly; much richer than a type library, metadata includes descriptions of an assembly and modules, classes, interfaces, methods, properties fields, events, global methods, and so forth).

26. **As to claim 20** (incorporating the rejection in claim 19), Lam discloses the method wherein said step of generating metadata information includes adding a reference to methods of the application assigned to components on the particular form (e.g., Sec. 2.3 – Metadata, 1st Para. – Metadata is machine-readable information about a resource, or “data about data”; Such information might include details on content, format, size, or other characteristics of a data source; In .NET®, metadata includes type definitions, version information, external assembly references, and other standardized information; Sec. 2.3.1, 1st Para. – Just as type libraries are C++ header files on steroids, metadata is a type library on steroids; in .NET® metadata is a common mechanism or dialect that the .NET® runtime, compilers, and tools can all use. Microsoft .NET® uses metadata to describe all types that are used and exposed by a particular .NET® assembly; much richer than a type library, metadata includes descriptions of an assembly and modules, classes, interfaces, methods, properties fields, events, global methods, and so forth).

27. **As to claim 21** (incorporating the rejection in claim 1), Albahari discloses the method further comprising: persisting state of the particular form, enabling the particular form to be recreated at runtime (e.g., Chapter 35 –

System.Reflection, 4th Para. – Serialization takes an existing object instance, uses reflection to suck out the object's state, transforms it into a binary representation, and stores that representation (a stream of bytes) to some source, such as a file on disk, a socket, a binary column in a database, and son on; Later, serialization can also take that same steam of bytes and re-hydrate the serialized object back into existence).

28. **As to claim 22** (incorporating the rejection in claim 21), Albahari discloses the method wherein said persisting step includes persisting user input on the particular form (e.g., Chapter 35 – System.Reflection, 2nd Para. – reflection offers up a number of possible approaches to user. Introspection is the act of using the reflection APIs to discover information about a component assembly at runtime without any prior (compile-tike) knowledge of it. This approach was first popularized by tools Visual Basic and numerous Java® IDEs that offered GUI-based construction of visual interfaces; the third-party component was dripped into some well-known location, and the IDE "discovered" it and offered it on a tool bar the next time the IDE was started; 4th Para. – Serialization takes an existing object instance, uses reflection to suck out the object's state, transforms it into a binary representation, and stores that representation (a stream of bytes) to some source, such as a file on disk, a socket, a binary column in a database, and son on; Later, serialization can also take that same steam of bytes and re-hydrate the serialized object back into existence).

29. **As to claim 23** (incorporating the rejection in claim 1), please refer to above claim **1**.

30. **As to claim 24** (incorporating the rejection in claim 1), Albahari discloses a downloadable set of processor-executable instructions for performing the method (e.g., Chapter 23 – C#® Development Tools, GacUtil.exe – [Description] – allows you to install, uninstall, and list the contents of the Global Assembly Cache (GAC), [Commands] - /ldl – lists all assemblies that were downloaded from the network).

31. Claims 8-11 and 25-48 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lam in view of Albahari, and further in view of Charles P. Jazdzewski (Pat. No. 6,002,867) (hereinafter 'Jazdzewski')

32. **As to claim 8** (incorporating the rejection in claim 1), Lam and Albahari do not explicitly disclose the method wherein said constructing step includes displaying a component palette including components which the user can select for placement on the particular form.

However, in an analogous art of Development System with Methods Providing Visual Form Inheritance, Jazdzewski discloses the method wherein said constructing step includes displaying a component palette including components which the user can select for placement on the particular form (e.g., Figs. 6A-6E; Col. 4, Lines 20-24, Figs. 6A-E are bitmap screenshots illustrating a preferred interface whereby a user can customize an inherited component on a descendant form..; Col. 2, Lines 11-15 – such environments are characterized by an Integrated Development Environment (IDE) providing a from …, a tool palette [with objects which the user can drag and drop on forms]….).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Jazdzewski into the Lam- Albahari's system to further provide the method wherein said constructing step includes displaying a component palette including components which the user can select for placement on the particular form in Lam- Albahari system.

The motivation is that it would further enhance the Lam- Albahari's system by taking, advancing and/or incorporating Jazdzewski's system which offers significant advantages that form inheritance allows the user to create a library of standard form templates, either within a single application or across an entire suite of applications; any changes made to the ancestor form immediately appear in the descendant forms as once suggested by Jazdzewski (e.g., Abstract, Lines 9-17).

33. **As to claim 9** (incorporating the rejection in claim 8), Jazdzewski discloses the method further comprising: receiving user input for placing components selected from the palette on the particular form (e.g., Fig. 3, elements "392" – Form Object, "393" - Properties, and "394" – Events; Col. 7, Lines 28-37 – the inspector 391 comprises an object selector field 392, a properties page 393, and an events page 394).

34. **As to claim 10** (incorporating the rejection in claim 9), Albahari discloses the method wherein the type delegator tracks creation of components on the particular form in response to a user placing a component on the particular form (e.g., Fig. 35-2 – Exceptions, delegates, and attributes from System.Reflection – element of "TypeDelegator"; Chapter 35. – System.Reflection, 1st Par. – System.Reflection is the API that exposes the full-fidelity metadata of the .NET® environment to the programmer; In short, it permits complete access to compile-time data at runtime; Everything is available including fields, methods, constructors, properties, delegate types, and events).

35. **As to claim 11** (incorporating the rejection in claim 9), Albahari discloses the method wherein the type delegator persists information regarding components placed on the particular form, thereby enabling the components placed on the particular form to be recreated at runtime (e.g., Chapter 35 – System.Reflection, 4th Para. – Serialization takes an existing object instance, uses reflection to suck out the object's state, transforms it into a binary representation, and stores that representation (a stream of bytes) to some source, such as a file on disk, a socket, a binary column in a database, and son on; Later, serialization can also take that same steam of bytes and re-hydrate the serialized object back into existence).

36. **As to claim 25**, Lam discloses a development system for dynamically constructing a form responsive to user input under an object framework during development of an application, the system comprising:

- an ancestor class for representing the form under the object framework; a proxy module for creating a descendant class inheriting from the ancestor class in response to user input, dynamically generating methods of the descendant class, and constructing an instance of the descendant class under the object framework for representing a particular form in the development system (e.g., Fig. 8-2 – System.Windows.Forms Windows Controls Class Hierarchy; Sec. 8.2 – The System.Windows.Forms Namesapce, 3rd Para. – the System.Windows.Forms namespace provides common set of classes you can use and derive from to build Windows® Forms application; the classes and interfaces in this namespace allow you to construct and render the user-interface elements on a Windows® Form; Sec. 8.3.3 – Visual Inheritance, 2nd Par. – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes);

Lam does not explicitly disclose a type delegator for the descendant class for persisting user input on the particular form during development of the application.

However, in an analogous art of C#® in a Nutshell, Albahari discloses a type delegator for the descendant class for persisting user input on the particular form during development of the application (e.g., Fig. 35-2 – Exceptions, delegates, and attributes from System.Reflection – element of "TypeDelegator"; Chapter 35. – System.Reflection, 1st Par. – System.Reflection is the API that exposes the full-fidelity metadata of the .NET® environment to the programmer; In short, it permits complete access to compile-time data at runtime; Everything is available including fields, methods, constructors, properties, delegate types, and events).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Albahari into the Lam's system to further provide a type delegator for the descendant class for persisting user input on the particular form during development of the application in Lam system.

The motivation is that it would further enhance the Lam's system by taking, advancing and/or incorporating Albahari's system which offers significant advantages that reflection offers up a number of possible approaches to user; introspection is the act of using the reflection APIs to discover information about a component assembly (and its constituent types) at runtime without any prior (compile-time) knowledge of it as once suggested by Albahari (e.g., Chapter 35. – System.Refection, 2nd Par.).

Lam and Albahari do not explicitly disclose a module for displaying the particular form in a user interface of the development system based on the descendant class and the persisted user input.

However, in an analogous art of Development System with Methods Providing Visual Form Inheritance, Jazdzewski discloses a module for displaying the particular form in a user interface of the development system based on the descendant class and the persisted user input (e.g., Col. 8, Lines 33-49 – nevertheless, the persistent image makes it appear to be virtual [because the most derived would be called]; accordingly, the call can be treaded as an "inherited" call; since a persistent steam exists, the system need only write out those properties that have changed [which are recorded in the type information]; there, the only information written out is that which is different from the default instantiation for the object; if the ancestor is viewed as a set of default values, then the descendant can be saved by only streaming out those values or properties which have changed; since a descendant does not save out values which the descendant has not changed, any properties of the ancestor which undergo change are simply passed through to the descendant [since the descendant does not store its own value for these]; at the descendant, one can override on a property-by-property basis values for the ancestor, but still get those values which have not been overridden).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Jazdzewski into the Lam- Albahari's system to further provide a module for displaying the particular form in a user interface of the development system based on the descendant class and the persisted user input in Lam-Albahari system.

The motivation is that it would further enhance the Lam- Albahari's system by taking, advancing and/or incorporating Jazdzewski's system which offers significant advantages that form inheritance allows the user to create a library of standard form templates, either within a single application or across an entire suite of applications; any changes made to the ancestor form immediately appear in the descendant forms as once suggested by Jazdzewski (e.g., Abstract, Lines 9-17).

37. **As to claim 26** (incorporating the rejection in claim 25), please refer to claim **2** above.

38. **As to claim 27** (incorporating the rejection in claim 25), please refer to claim **5** above.

39. **As to claim 28** (incorporating the rejection in claim 25), please refer to claim **3** above.

40. **As to claim 29** (incorporating the rejection in claim 28), Jazdzewski discloses the system wherein the descendant class inherits a set of components provided by the ancestor class for representing components that may be placed on the particular form (e.g., Abstract, Lines 1-4 – a visual development system is described which allows a user to derive forms from other "ancestor" forms, inheriting their components, properties, and code as a starting point for one's own forms).

41. **As to claim 30** (incorporating the rejection in claim 28), please refer to claim **6** above.

42. **As to claim 31** (incorporating the rejection in claim 28), Jazdzewski discloses the system wherein the form includes a component palette comprising components which the user can select (e.g., Fig. 3; Col. 6, Lines 36-37 – Fig. 3 illustrates an application development environment, which is provided by Delphi®; Lines 41-46 – as show, the programming environment 360 comprises an main window 361, a form 371, a code editor window 381, and an object manager or "inspector" window 391; the main window 361 itself comprises main menu 361, tool bar buttons 363, and component palette 364; main menu 362 lists user-selectable commands, in a conventional manner).

43. **As to claim 32** (incorporating the rejection in claim 31), please refer to claim **22** above.

44. **As to claim 33** (incorporating the rejection in claim 32), please refer to claim **21** above.

45. **As to claim 34** (incorporating the rejection in claim 25), please refer to claim **12** above.

46. **As to claim 35** (incorporating the rejection in claim 34), please refer to claim **13** above.

47. **As to claim 36** (incorporating the rejection in claim 35), please refer to claim **14** above.

48. **As to claim 37** (incorporating the rejection in claim 35), please refer to claim **15** above.

49. **As to claim 38** (incorporating the rejection in claim 25), please refer to claim **16** above.

50. **As to claim 39** (incorporating the rejection in claim 38), please refer to claim **17** above.

51. **As to claim 40** (incorporating the rejection in claim 39), Lam discloses the system wherein the proxy module generates intermediate language instructions using classes for generating intermediate language instructions provided by the object framework (e.g., Sec. 2.5 – Intermediate Language (IL), 3rd Para – Microsoft® calls its own language-abstraction layer the Common Intermediate Language (CIL); Similar bytecode, IL supports all object-oriented features, including data abstraction, inheritance, polymorphism and useful concepts such as exceptions and events; any .NET® language may be converted into IL, so

.NET® supports multiple languages and perhaps multiple platforms in the future [as long as the target platforms have a CLR]).

52. **As to claim 41** (incorporating the rejection in claim 25), please refer to claim **18** above.

53. **As to claim 42** (incorporating the rejection in claim 25), please refer to claim **19** above.

54. **As to claim 43** (incorporating the rejection in claim 42), please refer to claim **20** above.

55. **As to claim 44** (incorporating the rejection in claim 43), Lam discloses the system wherein said metadata information and the descendant class are used to reconstruct the form as part of the application at runtime (e.g., Sec. 2.3 – Metadata, 1st Para. – Metadata is machine-readable information about a resource, or "data about data"; Such information might include details on content, format, size, or other characteristics of a data source; In .NET®, metadata includes type definitions, version information, external assembly references, and other standardized information; Sec. 2.3.1, 1st Para. – Just as type libraries are C++ header files on steroids, metadata is a type library on steroids; in .NET® metadata is a common mechanism or dialect that the .NET® runtime, compilers, and tools can all use. Microsoft .NET® uses metadata to describe all types that

are used and exposed by a particular .NET® assembly; much richer than a type library, metadata includes descriptions of an assembly and modules, classes, interfaces, methods, properties fields, events, global methods, and so forth).

56. **As to claim 45** (incorporating the rejection in claim 25), Lam discloses the system wherein the form comprises a form open on a visual design surface of the development system (e.g., 8.3.3 – Visual Inheritance, 2nd Para. – with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from you rom class to create other for classes).

57. **As to claim 46** (incorporating the rejection in claim 25), Albahari discloses the system further comprising: a persisting mechanism for persisting state of the form (e.g., Chapter 35 – System.Reflection, 4th Para. – Serialization takes an existing object instance, uses reflection to suck out the object's state, transforms it into a binary representation, and stores that representation (a stream of bytes) to some source, such as a file on disk, a socket, a binary column in a database, and son on; Later, serialization can also take that same steam of bytes and re-hydrate the serialized object back into existence).

58. **As to claim 47** (incorporating the rejection in claim 46), Albahari discloses the system wherein the persisting mechanism persists user input on the form

during development of the application in the development system (e.g., Chapter 35 – System.Reflection, 4th Para. – Serialization takes an existing object instance, uses reflection to suck out the object's state, transforms it into a binary representation, and stores that representation (a stream of bytes) to some source, such as a file on disk, a socket, a binary column in a database, and son on; Later, serialization can also take that same steam of bytes and re-hydrate the serialized object back into existence).

59. **As to claim 48** (incorporating the rejection in claim 46), Albahari discloses the system wherein the persisting mechanism enables the form to be recreated at runtime as part of the application (e.g., Chapter 35 – System.Reflection, 4th Para. – Serialization takes an existing object instance, uses reflection to suck out the object's state, transforms it into a binary representation, and stores that representation (a stream of bytes) to some source, such as a file on disk, a socket, a binary column in a database, and son on; Later, serialization can also take that same steam of bytes and re-hydrate the serialized object back into existence).

Conclusion

60. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

- Adams et al., *.NET Windows® Forms in a Nutshell, March 2003, O'Reilly®.*

61. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

[illegible]

TUAN DAM
SUPERVISORY PATENT EXAMINER

BCW [illegible]

August 27, 2007